DW_arb_dp

Arbiter with Dynamic Priority Scheme

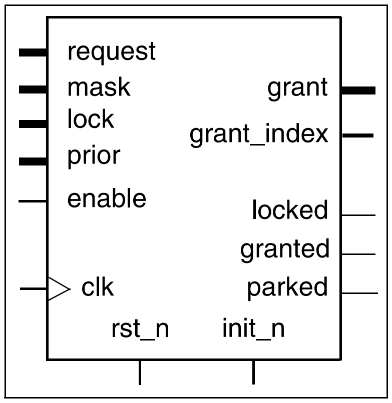
Version, STAR, and myDesignWare Subscriptions: IP Directory

DesignWare minPower Building Block

Features and Benefits

Revision History

- Parameterizable number of clients
- Programmable mask for all clients
- Park feature - default grant when no requests are pending
- Lock feature - ability to lock the currently granted client
- Registered/unregistered outputs
- Includes a low-power implementation that has power benefits from minPower optimization (for details, see Table 1-3 on page 3)



Applications

- Control application
- Networking
- Bus interfaces

Description

DW_arb_dp implements a parameterized, synchronous arbiter in which the priorities of requesting clients can be dynamically programmed. In this scheme, each of the n clients of the arbiter can be programmed with a ceil($\log_2 n$) bit `prior` input that is used to decide which of the requesting clients is issued the grant signal.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Asynchronous reset for all registers (active low)
init_n	1 bit	Input	Synchronous reset for all registers (active low)
enable	1 bit	Input	Enables clocking (active high)
request	*n* bits	Input	Input request from clients
prior	n*ceil($\log_2 n$) bits	Input	Priority vector from the clients of the arbiter

Table 1-1 Pin Description (Continued)

Pin Name	Width	Direction	Function
lock	*n* bits	Input	Signal to lock the grant to the current request ■ For *lock* (*i*) = 1, the arbiter is locked to the *request* (*i*) if it is currently granted. ■ For lock (*i*) = 0 the lock on the arbiter is removed.
mask	*n* bits	Input	Input to mask specific clients ■ For *mask* (*i*) = 1, *request* (*i*) is masked. ■ For *mask* (i) = 0 the mask on *request* (i) is removed.
parked	1 bit	Output	Flag to indicate that there are no requesting clients and the grant of resources has defaulted to the client designated by *park_index*
granted	1 bit	Output	Flag to indicate that the arbiter has issued a grant to one of the requesting clients
locked	1 bit	Output	Flag to indicate that the arbiter is locked by a client
grant	*n* bits	Output	Grant output
grant_index	$\log_2 n$ bits	Output	Index of the requesting client that has been currently granted or the client designated by *park_index* in *park_mode*

Table 1-2 Parameter Description

Parameter	Values	Description
n	2 to 32 Default: 4	Number of arbiter clients
park_mode	0 or 1 Default: 1	■ 1: Includes logic to enable parking when no clients are requesting ■ 0: Contains no logic for parking
park_index	0 to *n*–1 Default: 0	Index of the client used for parking
output_mode	0 or 1 Default: 1	■ 1: Includes registers at the outputs (see Figure 1-2 on page 5) ■ 0: Contains no output registers (see Figure 1-3 on page 5)

Table 1-3 Synthesis Implementations

Implementation Name	Function	License Feature Required
rtl	Synthesis model	DesignWare
lpwr[a]	Low Power Synthesis model	■ DesignWare (P-2019.03 and later) ■ DesignWare-LP (before P-2019.03)

a. Requires that you enable minPower; for details, see "Enabling minPower" on page 11.
When minPower is enabled, the lpwr implementation is always chosen during synthesis.

Table 1-4 Simulation Models

Model	Function
DW05.DW_ARB_DP_SIM_CFG	Design unit name for VHDL simulation
dw/dw05/DW_arb_dp_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_arb_dp.v	Verilog simulation model source code

Table 1-5 Arbiter Status Flags

Flag	Characteristic	Description
parked	If `parked` is active, there are no active requests at the input of the arbiter.	The `parked` output, active high, indicates that grant of the resources has defaulted to the client defined by *park_index* in *park_mode* = 1. In *park_mode* = 0, this flag does not exist.
granted	If `granted` is active, there is at least one active request at the input of the arbiter.	The `granted` output, active high, indicates that the grant of resources is to one of the actively requesting inputs.
locked	If `locked` is active, the current grant and the corresponding lock signal must be active.	The `locked` output, active high, indicates that the currently granted client has locked out all other clients.

The dynamic priority programming of DW_arb_dp comes from concatenated numeric priority values on the input port, `prior`. Clients programmed with lower numeric priority values have higher arbitration priority. Thus, when the bits of the `prior` input for a particular client are set to all zeros, that client is part of the highest priority group. For a diagram that explains the priority vector, see Figure 1-1 on page 4.

To determine priority when two or more arbiter clients are programmed with the same priority value, the DW_arb_dp uses the index of `request` inputs. The client connected to `request[0]` has the highest priority, while the client connected to `request[`*n*`-1]` has the lowest priority.

At the extreme, if all clients are programmed with the same numeric priority value, then DW_arb_dp operates exactly like the static priority component, DW_arb_sp, where priority is statically set by client position (which considers client 0 as having highest arbitration priority).

The lock feature enables a client, despite requests from other clients, to have an exclusive grant for the duration of the corresponding lock input. After a client receives the grant, it can lock out other clients from the arbitration process by setting the corresponding lock input.

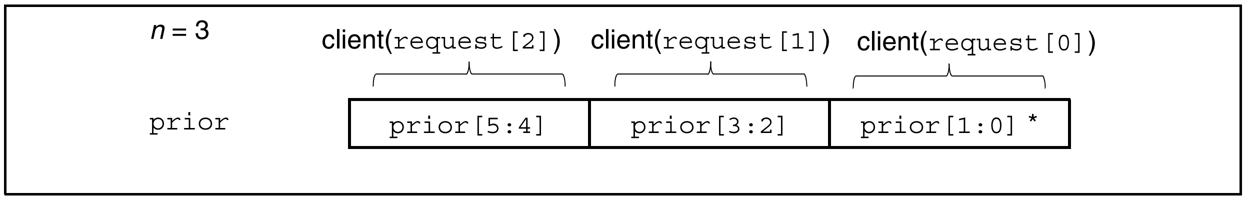
The park feature allows the resources to be granted to a designated client defined by the *park_index* parameter when there are no active requests pending. The *park_mode* parameter enables/disables the parking feature.

By setting the desired bits of the `mask` input, the corresponding clients can be masked off from consideration for arbitration. The mask on a client remains active until the corresponding mask input for the client is reset.

All the input requests from the arbiter clients are assumed to be synchronized to the arbiter clock signal `clk`. The arbiter provides several flags, `locked`, `granted`, and `parked`, to indicate the status of the arbiter.

The width of the `prior` input vector is n `* ceil(`$\log_2 n$`)`, which is formed by concatenating the `prior` values of the n arbiter clients (see Figure 1-1.) Each of the clients can be programmed with a priority level from 0 (highest priority) to n–1 (lowest priority). Two or more clients can be programmed with the same priority level. As mentioned earlier, in such cases, the index of connection of the clients to the arbiter is used to break the tie in the arbitration.

Figure 1-1 Priority Vector



* Each client `prior` slice is “ceil(log2*n*)” bits wide

The mask, park and lock features add flexibility to the arbiter. The parking of grant to a designated client saves an arbitration cycle and the parked client can lock the grant without issuing a request to the arbiter.

In cases where the number of clients (*n)* is NOT a power of 2 (2, 4, 16, or 32), the priority of a client should be programmed only with a value from 0 to n–1. The possible values from n to $2^{\mathrm{ceil}(\log_2 n)}$ are illegal, and if used, the arbiter outputs incorrect results.

Figure 1-2 Block Diagram of DW_arb_dp Arbiter, output_mode =1

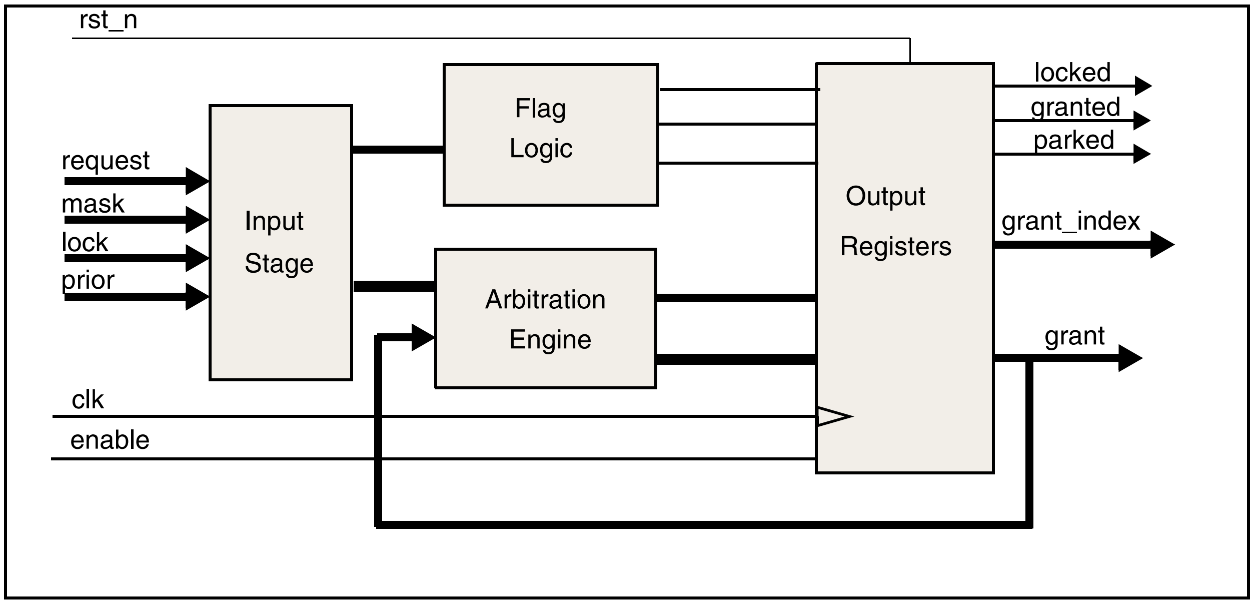
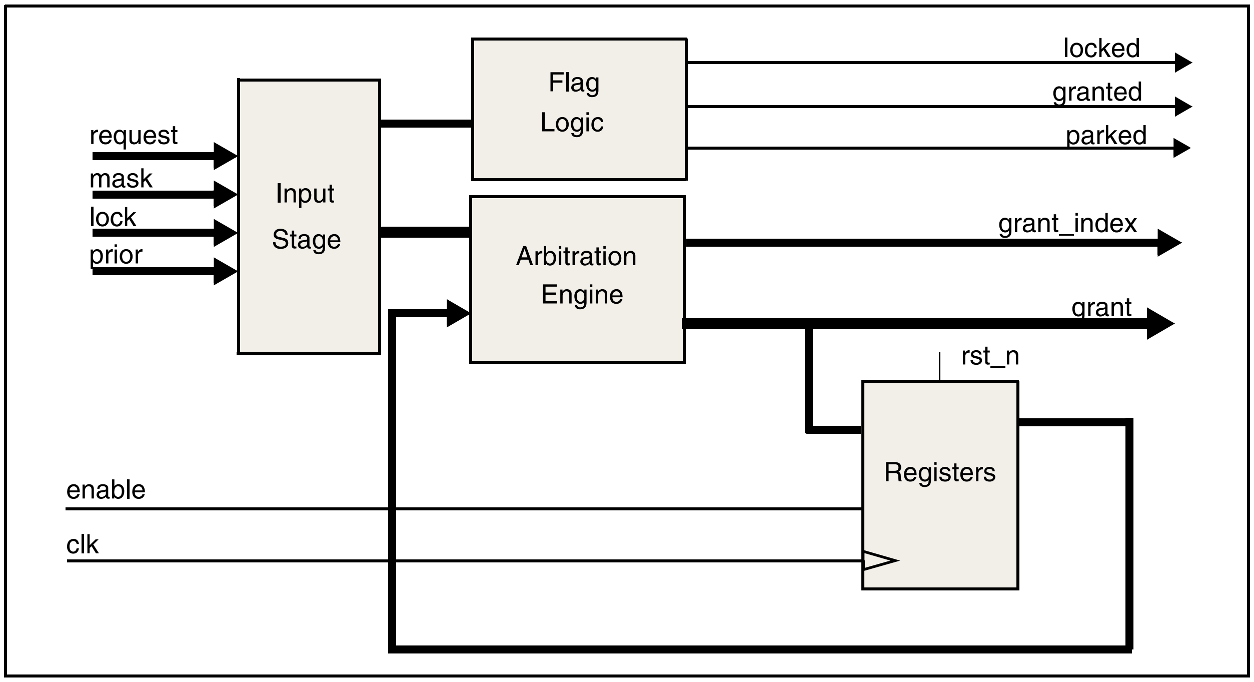


Figure 1-3 Block Diagram of DW_arb_dp Arbiter, output_mode = 0



Low Power Implementation

This component provides low power (minPower) benefits when the "lpwr" implementation is chosen, as described in Table 1-3 on page 3. Effectiveness of low power design depends on the use of the `-gate_clock` option to `compile_ultra` command.

Suppressing Warning Messages During Verilog Simulation

The Verilog simulation model includes macros that allow you to suppress warning messages during simulation.

To suppress all warning messages for all DWBB components, define the DW_SUPPRESS_WARN macro in either of the following ways:

- Specify the Verilog preprocessing macro in Verilog code:

  ```
  `define DW_SUPPRESS_WARN
  ```

- Or, include a command line option to the simulator, such as:

  `+define+DW_SUPPRESS_WARN` (which is used for the Synopsys VCS simulator)

The warning messages for this model include the following:

- If values other than 1 or 0 are present on a clock port, the following message is displayed:

  ```
  WARNING: <instance_path>.<clock_name>_monitor:
      at time = <timestamp>, Detected unknown value, x, on <clock_name> input.
  ```

  To suppress only this warning message for all DWBB components, use the following macro:

  - Define the DW_DISABLE_CLK_MONITOR macro. You can define this macro in the following ways:
    - Specify the Verilog preprocessing macro in Verilog code:

      ```
      `define DW_DISABLE_CLK_MONITOR
      ```

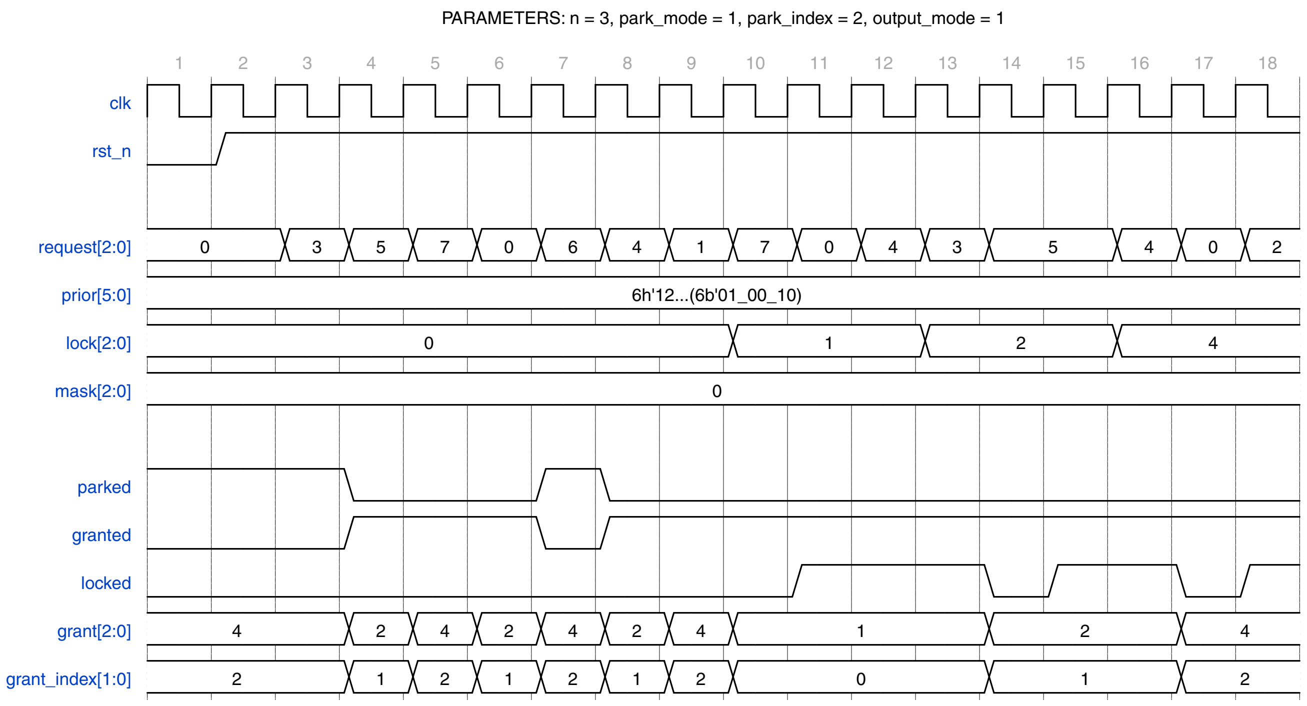
    - Or, include a command line option to the simulator, such as:

      `+define+DW_DISABLE_CLK_MONITOR` (which is used for the Synopsys VCS simulator)

  This message is also suppressed using the DW_SUPPRESS_WARN macro explained earlier.

Timing Waveforms

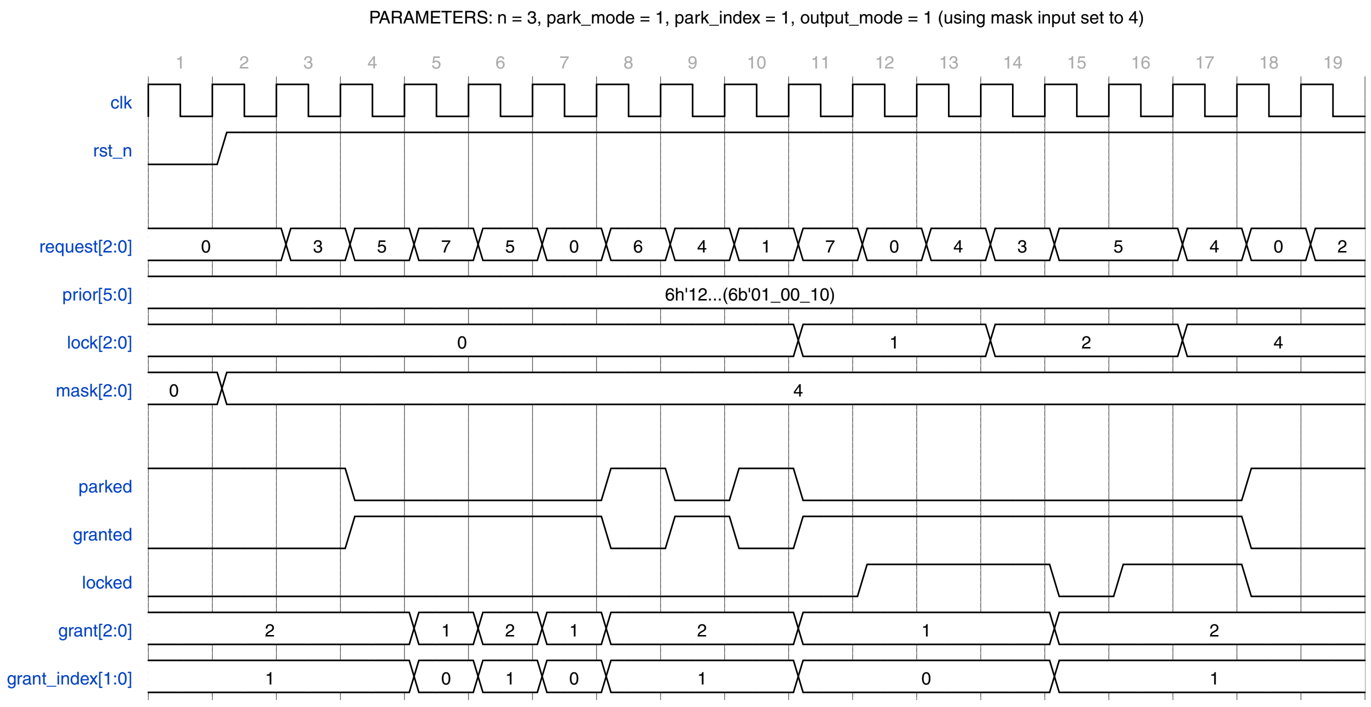
Figure 1-4 Waveform 1



As shown in Figure 1-4, the vector for `prior` in the above mentioned case of 6h'12 (6b'01_00_10) translates to priority value client[2] = 1, priority value client[1] = 0 (highest priority), and priority value client[0] = 2 (lowest priority).

Figure 1-5 shows masking behavior. Any client can be masked off by setting the corresponding mask bit. Masked clients are not considered for the arbitration. If mask bits are set and none of the non-masked clients are actively requesting, the arbiter is parked to the designated client defined by *park_index*. In the non-locked state of the arbiter, setting the mask bit of the currently granted client effectively invalidates the request from the client. In the following cycle, the current grant is de-asserted, and based on the current unmasked requests from other clients, a new client is generated. However, when a client has locked the arbiter, setting the mask bit of any client has no effect on the current grant.

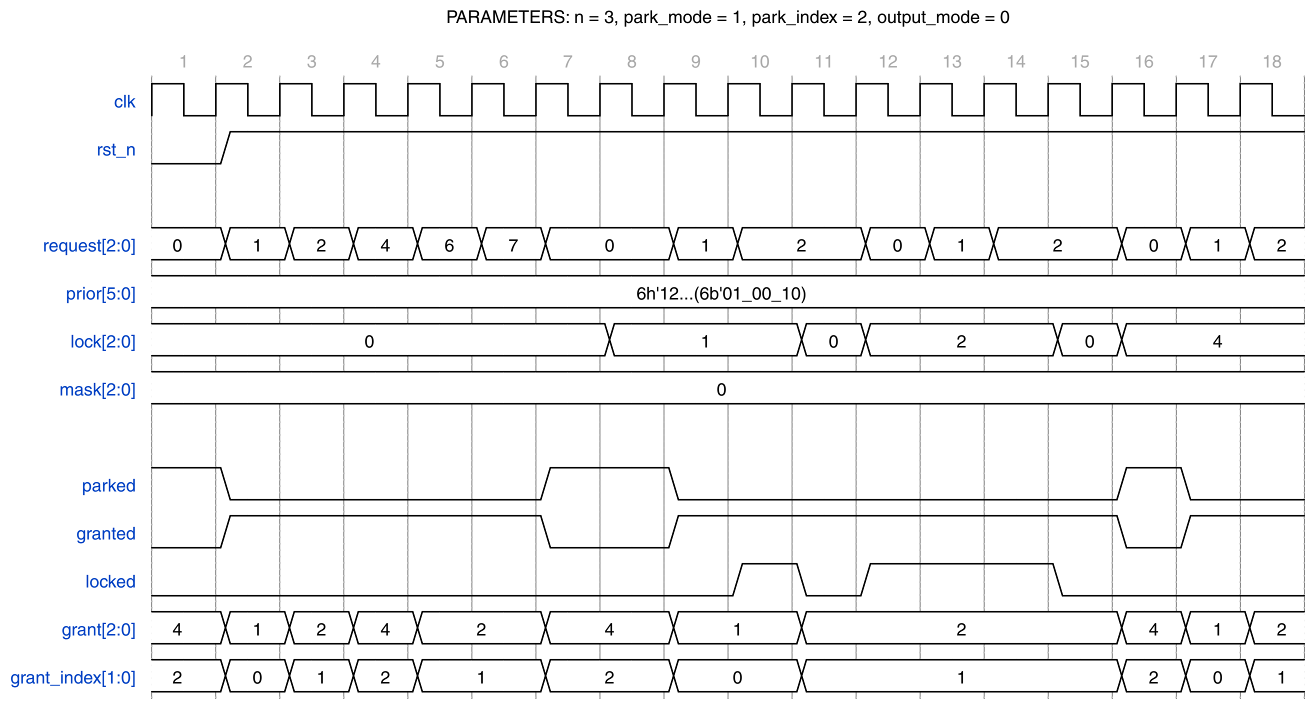
Figure 1-5 Waveform 2



As shown in Figure 1-5, the priorities of the three clients of the arbiter in the above mentioned case are client[0] = 2 (lowest priority), client[1] = 0 (highest priority), and client[2] = 1.

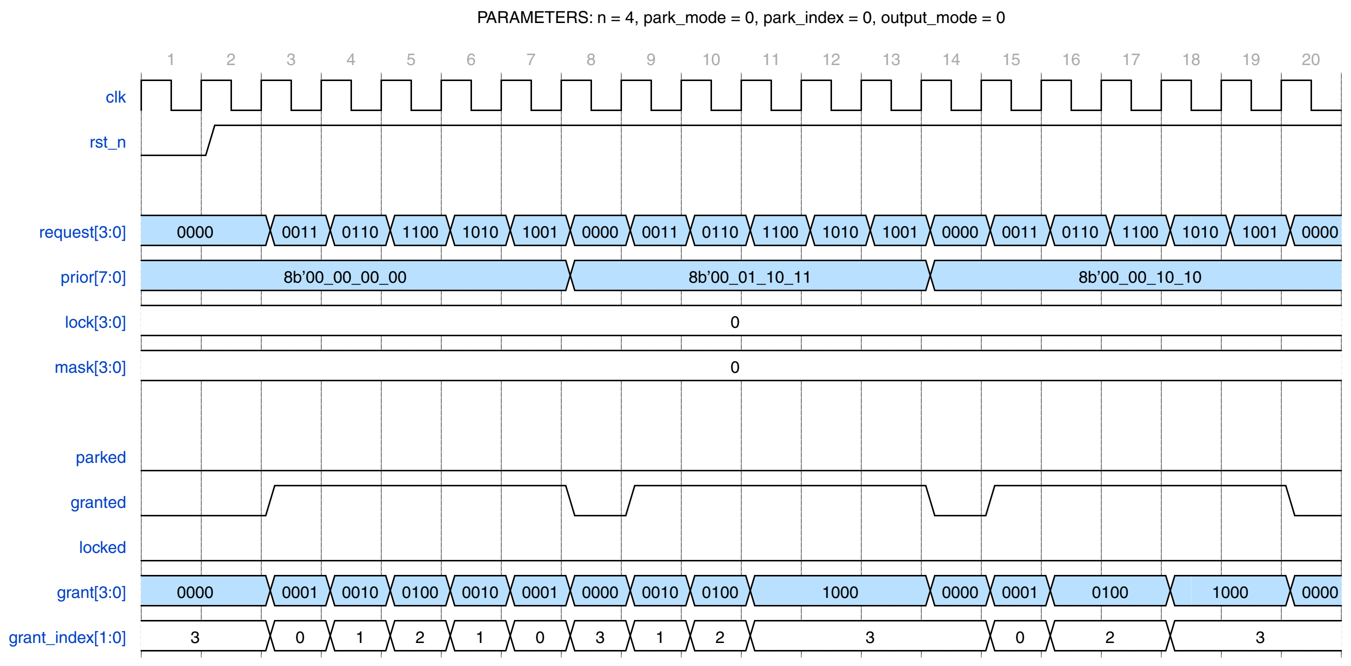
Figure 1-6 shows another case of unmasked priority behavior for DW_arb_dp.

Figure 1-6 Waveform 3



In Figure 1-7, a sequence of input requests is repeated for three different values placed on the dynamic priority control input, `prior`.

Figure 1-7 Waveform 4



The sequence is first applied when `prior` is set to all zeros, which instructs DW_arb_dp that all inputs have the same "dynamic priority" so any arbitration of multiple bidders resorts to the underlying inherent static priority where `request`[0] has the highest priority and `request`[n-1] has the lowest. Under this condition (that is, when all clients are assigned the same dynamic priority), DW_arb_dp functions the same as DW_arb_sp. Simply put, under this situation, the right-most '1' on the (binary represented) `request` input is granted the bid.

The second time the input `request` sequence is applied, the dynamic priority input, `prior`, is set to 8b'00 01 10 11. Because an assigned priority value of all zeros is the highest priority and an all-ones value represents the lowest, this totally reverses the priority with respect to the DW_arb_sp. That is, instead of `request`[0] having the highest priority, this setting causes `request`[3] to have the highest priority. Similarly, this value on the `prior` input causes `request`[0] to have the lowest priority (as compared to DW_arb_sp, where `request`[3] has the lowest priority). Simply put, with the `prior` input set the way it is in clock cycles 8 through 13, the left most '1' on the (binary represented) `request` input is granted the bid.

In the third input request sequence, `prior` is set to 8b'00 00 10 10. Here, client request on `request`[0] and `request`[1] are set to the same dynamic priority (binary 10) and client request on `request`[2] and `request`[3] are also set to the same dynamic priority (binary 00). So, the two inputs `request`[2] and `request`[3] have a higher dynamic priority than `request`[0] and `request`[1]. Thus, as long as there is a bid on either `request`[2] or `request`[3], other input requests are shut out. But, when there are multiple bidders with the same dynamic priority programmed through the `prior` input, final arbitration resorts to the fall-back (static) priority scheme. In Figure 1-7, this is illustrated in clock cycles 15 and 17. In clock cycle 15, `request`[0] and `request`[1] come at the same time while they have the same dynamic priority and the fall-back priority gives `request`[0] the higher static priority.

Enabling minPower

You can instantiate this component without enabling minPower, but to achieve power savings from the low-power implementation "lpwr" (see Table 1-3 on page 3), you must enable minPower optimization, as follows:

- Design Compiler
  - Version P-2019.03 and later:

    ```
    set power_enable_minpower true
    ```
  - Before version P-2019.03 (requires the DesignWare-LP license feature):

    ```
    set synthetic_library {dw_foundation.sldb dw_minpower.sldb}
    set link_library {* $target_library $synthetic_library}
    ```
- Fusion Compiler

  Optimization for minPower is enabled as part of the total_power metric setting. To enable the total_power metric, use the following:

  ```
  set_qor_strategy -stage synthesis -metric total_power
  ```

Related Topics

- Application Specific – Control Logic Overview
- DesignWare Building Block IP User Guide

HDL Usage Through Component Instantiation - VHDL

```
library IEEE,DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DWpackages.all;
use DWARE.dw_foundation_comp.all;

entity DW_arb_dp_inst is
      generic (
        inst_n : NATURAL := 4;
        inst_park_mode : NATURAL := 1;
        inst_park_index : NATURAL := 0;
        inst_output_mode : NATURAL := 1
        );
      port (
        inst_clk : in std_logic;
        inst_rst_n : in std_logic;
        inst_init_n : in std_logic;
        inst_enable : in std_logic;
        inst_request : in std_logic_vector(inst_n-1 downto 0);
      inst_prior : in std_logic_vector(bit_width(inst_n)*inst_n-1 downto 0);
        inst_lock : in std_logic_vector(inst_n-1 downto 0);
        inst_mask : in std_logic_vector(inst_n-1 downto 0);
        parked_inst : out std_logic;
        granted_inst : out std_logic;
        locked_inst : out std_logic;
        grant_inst : out std_logic_vector(inst_n-1 downto 0);
       grant_index_inst : out std_logic_vector(bit_width(inst_n)-1 downto 0)
        );
    end DW_arb_dp_inst;


architecture inst of DW_arb_dp_inst is

begin

    -- Instance of DW_arb_dp
    U1 : DW_arb_dp
    generic map (
          n => inst_n,
          park_mode => inst_park_mode,
          park_index => inst_park_index,
          output_mode => inst_output_mode
          )
    port map (
          clk => inst_clk,
          rst_n => inst_rst_n,
          init_n => inst_init_n,
          enable => inst_enable,
          request => inst_request,
          prior => inst_prior,
          lock => inst_lock,
          mask => inst_mask,
          parked => parked_inst,
          granted => granted_inst,
          locked => locked_inst,
          grant => grant_inst,
          grant_index => grant_index_inst
          );


end inst;

-- pragma translate_off
configuration DW_arb_dp_inst_cfg_inst of DW_arb_dp_inst is
  for inst
  end for; -- inst
end DW_arb_dp_inst_cfg_inst;
-- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_arb_dp_inst( inst_clk, inst_rst_n, inst_init_n, inst_enable, inst_request,
          inst_prior, inst_lock, inst_mask, parked_inst, granted_inst,
          locked_inst, grant_inst, grant_index_inst );

parameter inst_n = 4;
parameter inst_park_mode = 1;
parameter inst_park_index = 0;
parameter inst_output_mode = 1;

`define bit_width_n 2// bit_width_n is set to ceil(log2(n))

input inst_clk;
input inst_rst_n;
input inst_init_n;
input inst_enable;
input [inst_n-1 : 0] inst_request;
input [(`bit_width_n*inst_n-1) : 0] inst_prior;
input [inst_n-1 : 0] inst_lock;
input [inst_n-1 : 0] inst_mask;
output parked_inst;
output granted_inst;
output locked_inst;
output [inst_n-1 : 0] grant_inst;
output [`bit_width_n-1 : 0] grant_index_inst;

    // Instance of DW_arb_dp
    DW_arb_dp #(inst_n, inst_park_mode, inst_park_index, inst_output_mode) U1 (
                .clk(inst_clk),
                .rst_n(inst_rst_n),
                .init_n(inst_init_n),
                .enable(inst_enable),
                .request(inst_request),
                .prior(inst_prior),
                .lock(inst_lock),
                .mask(inst_mask),
                .parked(parked_inst),
                .granted(granted_inst),
                .locked(locked_inst),
                .grant(grant_inst),
                .grant_index(grant_index_inst) );

endmodule
```

Revision History

For notes about this release, see the *DesignWare Building Block IP Release Notes*.

For lists of both known and fixed issues for this component, refer to the STAR report.

For a version of this datasheet with visible change bars, click here.

Date	Release	Updates
July 2020	DWBB_201912.5	■ Adjusted content and title of "Suppressing Warning Messages During Verilog Simulation" on page 6 and added the DW_SUPPRESS_WARN macro
October 2019	DWBB_201903.5	■ Added "Disabling Clock Monitor Messages"
June 12, 2019	DWBB_201903.2	■ Amplified the explanation of the priority scheme by updating Figures 1-4 through Figure 1-6 and adding Figure 1-7 and accompanying text
June 2019	DWBB_201903.2	■ Updated the explanation of the priority scheme on page 3
March 2019	DWBB_201903.0	■ Clarified license requirements in Table 1-3 on page 3 ■ Added "Enabling minPower" on page 11 ■ Added this Revision History table and the document links on this page